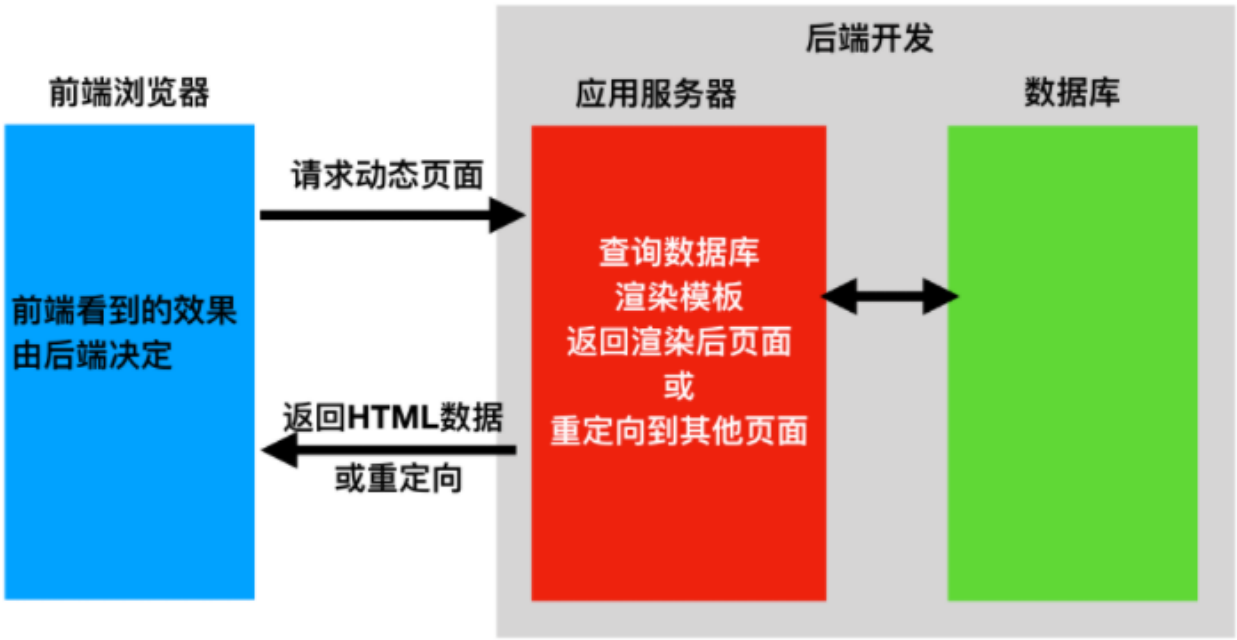
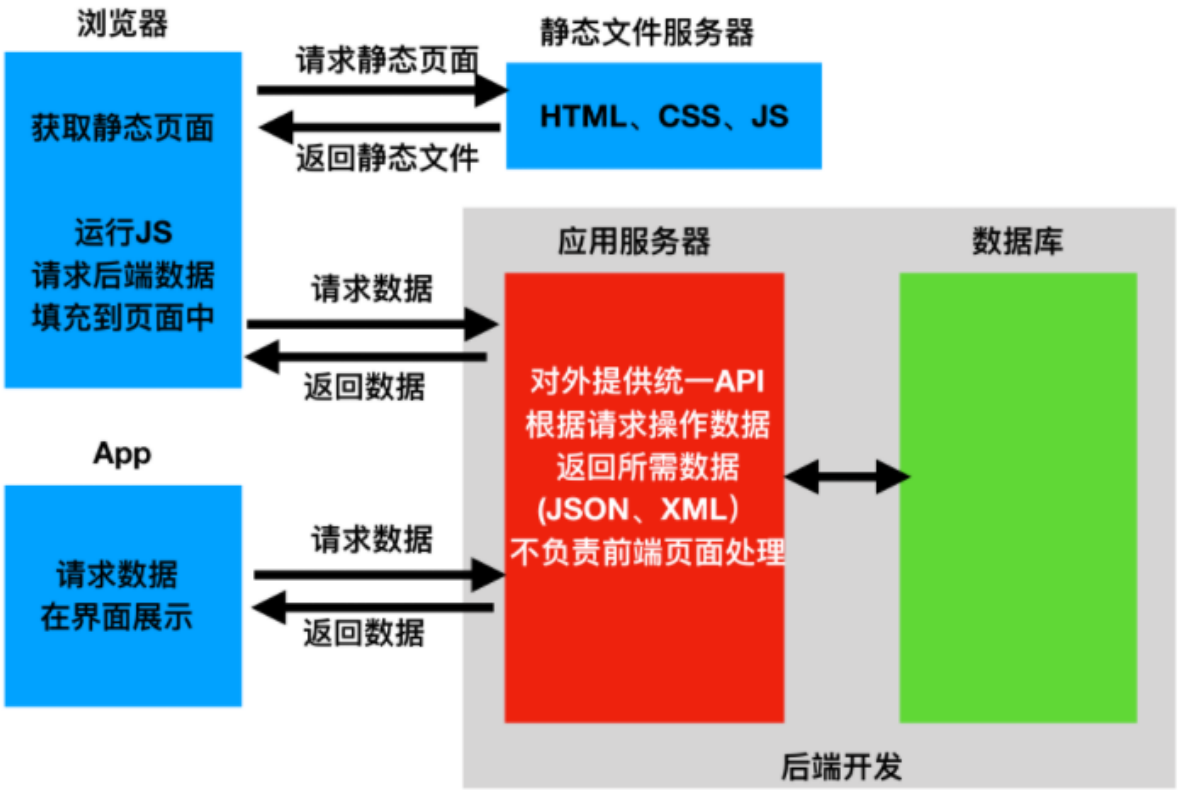
马士兵教育Python 全栈文档

第七章：Flask-RESTful 风格编程



在前后端不分离的应用模式中，前端页面看到的效果都是由后端控制，由后端渲染页面或重定向，也就是后端需要控制前端的展示，前端与后端的耦合度很高。

这种应用模式比较适合纯网页应用，但是当后端对接App时，App可能并不需要后端返回一个HTML网页，而仅仅是数据本身，所以后端原本返回网页的接口不再适用于前端App应用，为了对接App后端还需再开发一套接口。



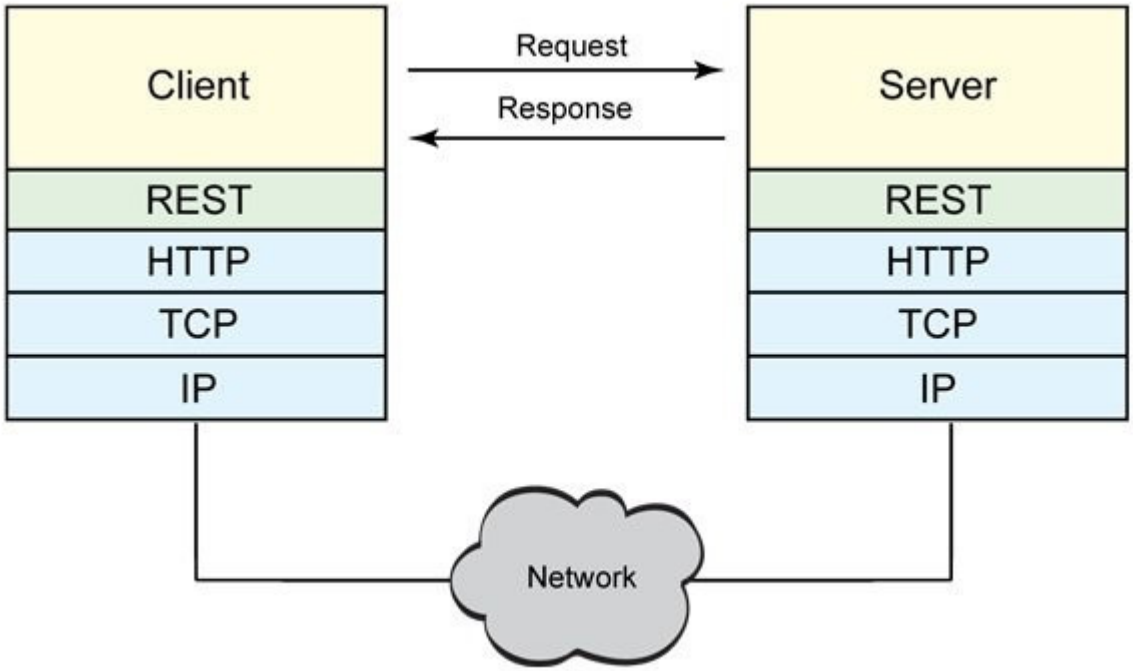
在前后端分离的应用模式中，后端仅返回前端所需的数据，不再渲染HTML页面，不再控制前端的效果。至于前端用户看到什么效果，从后端请求的数据如何加载到前端中，都由前端自己决定，网页有网页的处理方式，App有App的处理方式，但无论哪种前端，所需的数据基本相同，后端仅需开发一套逻辑对外提供数据即可。

在前后端分离的应用模式中，前端与后端的耦合度相对较低。

在前后端分离的应用模式中，我们通常将后端开发的每个视图都称为一个**接口**，或者**API**，前端通过访问接口来对数据进行增删改查。

一、RESTful定义

RESTFUL是一种网络应用程序的设计风格和开发方式，基于HTTP，可以使用XML格式定义或JSON格式定义。RESTFUL适用于移动互联网厂商作为业务使能接口的场景。



RESTFUL特点包括：

1、每一个URI代表1种资源；

2、客户端使用GET、POST、PUT、DELETE4个表示操作方式的动词对服务端资源进行操作：GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源；

3、通过操作资源的表现形式来操作资源；

4、资源的表现形式是XML或者 **JSON**；

5、客户端与服务端之间的交互在请求之间是无状态的，从客户端到服务端的每个请求都必须包含理解请求所必需的信息。

比如：(POST/PUT/GET/DELETE) - http://127.0.0.1:8080/AppName/rest?id=1

二、RESTful风格的软件架构

RESTful架构是对MVC架构改进后所形成的一种架构，通过使用事先定义好的接口与不同的服务联系起来。在RESTful架构中，客户端使用POST，DELETE，PUT和GET四种请求方式分别对指定的URL资源进行增删改查操作。因此，RESTful是通过URI实现对资源的管理及访问，具有扩展性强、结构清晰的特点。

RESTful架构将服务器分成前端服务器和后端服务器两部分，前端服务器为用户提供无模型的视图；后端服务器为前端服务器提供接口。浏览器向前端服务器请求视图，通过视图中包含的AJAX函数发起接口请求获取模型。

项目开发引入RESTful架构，利于团队并行开发。在RESTful架构中，将多数HTTP请求转移到前端服务器上，降低服务器的负荷，使视图获取后端模型失败也能呈现。但RESTful架构却不适用于所有的项目，当项目比较小时无需使用RESTful架构，项目变得更加复杂。

三、安装和使用

```
pip install flask-restful
```

1、普通使用

```
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorldResource(Resource):
    def get(self):
        return {'hello': 'world'}

        def post(self):
        return {'msg': 'hello laoxiao'}

api.add_resource(HelloWorldResource, '/')
```

注意：通过endpoint参数为路由起名：api.add_resource(HelloWorldResource, '/', endpoint='HelloWorld')

2、蓝图中使用

```
from flask import Flask, Blueprint
from flask_restful import Api, Resource

app = Flask(__name__)

user_bp = Blueprint('user', __name__)

user_api = Api(user_bp)

class HelloResource(Resource):
    def get(self):
        return {'msg': 'hello laoxiao'}

user_api.add_resource(HelloResource, '/users/hello)

app.register_blueprint(user_bp)
```

注意：如果蓝图里面有url_prefix，那么请求url = url_prefix + resource_url

四、Request和RequestParser类

1、RequestParser类

Flask-RESTful 提供了`RequestParser`类，用来帮助我们检验和转换请求数据。

```
from flask_restful import reqparse

parser = reqparse.RequestParser()
parser.add_argument('id', type=int, help='Rate cannot be converted', location='args')
parser.add_argument('name')
args = parser.parse_args()
```

使用步骤：

1. 创建`RequestParser`对象
2. 向`RequestParser`对象中添加需要检验或转换的参数声明
3. 使用`parse_args()`方法启动检验处理
4. 检验之后从检验结果中获取参数时可按照字典操作或对象属性操作

```
args.rate
或
args['rate']
```

2、参数说明

1、required

描述请求是否一定要携带对应参数，**默认值为False**

- True 强制要求携带

  若未携带，则校验失败，向客户端返回错误信息，状态码400

- False 不强制要求携带

  若不强制携带，在客户端请求未携带参数时，取出值为None

```
class DemoResource(Resource):
    def get(self):
        rp = RequestParser()
        rp.add_argument('a', required=False)
        args = rp.parse_args()
        return {'msg': 'data={}'.format(args.a)}
```

2、help

参数检验错误时返回的错误描述信息

```
rp.add_argument('a', required=True, help='missing a param')
```

3、action

描述对于请求参数中出现多个同名参数时的处理方式

- `action='store'` 保留出现的第一个， 默认
- `action='append'` 以列表追加保存所有同名参数的值

```
rp.add_argument('a', required=True, help='missing a param', action='append')
```

4、type

描述参数应该匹配的类型，可以使用python的标准数据类型string、int，也可使用Flask-RESTful提供的检验方法，还可以自己定义

- 标准类型

  ```
  rp.add_argument('a', type=int, required=True, help='missing a param',
  action='append')
  ```

- Flask-RESTful提供

  检验类型方法在`flask_restful.inputs`模块中

  - `url`
  - `regex(指定正则表达式)`

    ```
    from flask_restful import inputs
    rp.add_argument('a', type=inputs.regex(r'^\d{2}&'))
    ```

  - `natural` 自然数0、1、2、3...
  - `positive` 正整数 1、2、3...
  - `int_range(low ,high)` 整数范围

    ```
    rp.add_argument('a', type=inputs.int_range(1, 10))
    ```

  - `boolean`

- 自定义

  ```
  def mobile(mobile_str):
      """
      检验手机号格式
      :param mobile_str: str 被检验字符串
      :return: mobile_str
      """
      if re.match(r'^1[3-9]\d{9}$', mobile_str):
          return mobile_str
      else:
          raise ValueError('{} is not a valid mobile'.format(mobile_str))

  rp.add_argument('a', type=mobile)
  ```

5、location

描述参数应该在请求数据中出现的位置

```
# Look only in the POST body
```

```
parser.add_argument('name', type=int, location='form')

# Look only in the querystring
parser.add_argument('PageSize', type=int, location='args')

# From the request headers
parser.add_argument('User-Agent', location='headers')

# From http cookies
parser.add_argument('session_id', location='cookies')

# From json
parser.add_argument('user_id', location='json')

# From file uploads
parser.add_argument('picture', location='files')
```

也可指明多个位置

```
parser.add_argument('text', location=['headers', 'json'])
```

五、RESTful的响应处理

1、序列化数据

Flask-RESTful 提供了marshal工具，用来帮助我们将数据序列化为特定格式的字典数据，以便作为视图的返回值。

```
from flask_restful import Resource, fields, marshal_with

resource_fields = {
    'name': fields.String,
    'address': fields.String,
    'user_id': fields.Integer
}

class Todo(Resource):
    @marshal_with(resource_fields, envelope='resource')
    def get(self, **kwargs):
        return db_get_todo()
```

也可以不使用装饰器的方式

```
class Todo(Resource):
    def get(self, **kwargs):
        data = db_get_todo()
        return marshal(data, resource_fields)
```

示例

```
# 用来模拟要返回的数据对象的类
class User(object):
    def __init__(self, user_id, name, age):
        self.user_id = user_id
        self.name = name
        self.age = age

resoure_fields = {
        'user_id': fields.Integer,
        'name': fields.String
    }

class Demo1Resource(Resource):
    @marshal_with(resoure_fields, envelope='data1')
    def get(self):
        user = User(1, 'laoxiao', 12)
        return user

class Demo2Resource(Resource):
    def get(self):
        user = User(1, 'laoxiao', 12)
        return marshal(user, resoure_fields, envelope='data2')
```

2、定制返回的JSON格式

需求

想要接口返回的JSON数据具有如下统一的格式

```
{"message": "描述信息", "data": {要返回的具体数据}}
```

在接口处理正常的情况下， message返回ok即可，但是若想每个接口正确返回时省略message字段

```
class DemoResource(Resource):
    def get(self):
        return {'user_id':1, 'name': 'laoxiao'}
```

对于诸如此类的接口，能否在某处统一格式化成上述需求格式?

```
{"message": "OK", "data": {'user_id':1, 'name': 'laoxiao'}}
```

解决

Flask-RESTful的Api对象提供了一个representation的装饰器，允许定制返回数据的呈现格式

```
api = Api(app)

@api.representation('application/json')
def handle_json(data, code, headers):
    # TODO 此处添加自定义处理
    return resp
```

Flask-RESTful原始对于json的格式处理方式如下：

代码出处：`flask_restful.representations.json`

```
from flask import make_response, current_app
from flask_restful.utils import PY3
from json import dumps


def output_json(data, code, headers=None):
    """Makes a Flask response with a JSON encoded body"""

    settings = current_app.config.get('RESTFUL_JSON', {})

    # If we're in debug mode, and the indent is not set, we set it to a
    # reasonable value here.  Note that this won't override any existing value
    # that was set.  We also set the "sort_keys" value.
    if current_app.debug:
        settings.setdefault('indent', 4)
        settings.setdefault('sort_keys', not PY3)

    # always end the json dumps with a new line
    # see https://github.com/mitsuhiko/flask/pull/1262
    dumped = dumps(data, **settings) + "\n"

    resp = make_response(dumped, code)
    resp.headers.extend(headers or {})
    return resp
```

为满足需求，做如下改动即可

```
@api.representation('application/json')
def output_json(data, code, headers=None):
    """Makes a Flask response with a JSON encoded body"""

    # 此处为自己添加***************
    if 'message' not in data:
        data = {
            'message': 'OK from msb',
            'data': data
        }
    # *************************

    settings = current_app.config.get('RESTFUL_JSON', {})

    # If we're in debug mode, and the indent is not set, we set it to a
    # reasonable value here.  Note that this won't override any existing value
    # that was set.  We also set the "sort_keys" value.
    if current_app.debug:
        settings.setdefault('indent', 4)
        settings.setdefault('sort_keys', not PY3)

    # always end the json dumps with a new line
```

```
    # see https://github.com/mitsuhiko/flask/pull/1262
    dumped = dumps(data, **settings) + "\n"

    resp = make_response(dumped, code)
    resp.headers.extend(headers or {})
    return resp
```